CGL Reference

Graphics & Imaging > OpenGL

2007-06-28

Contents

CGL Reference

Framework:	OpenGL/OpenGL.h
Companion guide	OpenGL Programming Guide for Mac OS X
Declared in	CGLCurrent.h CGLTypes.h OpenGL.h

Overview

The CGL (Core OpenGL) API is lowest-level programming interface for the Apple implementation of OpenGL. CGL supports full screen OpenGL drawing and drawing to pixel buffers, which are a hardware-accelerated offscreen drawing location. Any Cocoa or Carbon application can use CGL to get the most direct access to system functionality. The Cocoa classes that support OpenGL and the AGL API are each built on top of CGL.

Functions by Task

Managing Pixel Format Objects

`CGLChoosePixelFormat` (page 8)
: Creates a pixel format object that satisfies the constraints of the specified buffer and renderer attributes.

`CGLDestroyPixelFormat` (page 17)
: Frees the memory associated with a pixel format object.

`CGLDescribePixelFormat` (page 14)
: Retrieves the values of an attribute associated with a pixel format object.

Managing Contexts

`CGLCreateContext` (page 11)
: Creates a CGL rendering context.

`CGLCopyContext` (page 10)
: Copies the specified state variables from one rendering context to another.

`CGLDestroyContext` (page 16)
: Frees the resources associated with a rendering context.

`CGLGetCurrentContext` (page 21)
    Returns the current rendering context.

`CGLSetCurrentContext` (page 27)
    Sets the specified rendering context as the current rendering context.

Getting and Setting Context Options

`CGLEnable` (page 19)
    Enables an option for a rendering context.

`CGLDisable` (page 18)
    Disables an option for a rendering context.

`CGLIsEnabled` (page 25)
    Reports whether an option is enabled for a rendering context.

`CGLSetParameter` (page 30)
    Sets the value of a rendering context parameter.

`CGLGetParameter` (page 22)
    Retrieves the value of a rendering context parameter.

Locking and Unlocking Contexts

`CGLLockContext` (page 26)
    Locks a CGL rendering context.

`CGLUnlockContext` (page 34)
    Unlocks a CGL rendering context.

Managing Drawable Objects

`CGLSetOffScreen` (page 29)
    Attaches a rendering context to an offscreen buffer.

`CGLGetOffScreen` (page 21)
    Retrieves an offscreen buffer and its parameters for a specified rendering context.

`CGLSetFullScreen` (page 28)
    Attaches a rendering context to its full-screen drawable object.

`CGLClearDrawable` (page 9)
    Disassociates a rendering context from any drawable objects attached to it.

`CGLFlushDrawable` (page 20)
    Copies the back buffer of a double-buffered context to the front buffer.

Managing Pixel Buffers

`CGLCreatePBuffer` (page 12)
    Creates a pixel buffer of the specified size, compatible with the specified texture target.

`CGLDescribePBuffer` (page 13)
: Retrieves information that describes the specified pixel buffer object.

`CGLDestroyPBuffer` (page 16)
: Releases the resources associated with a pixel buffer object.

`CGLGetPBuffer` (page 23)
: Retrieves a pixel buffer and its parameters for a specified rendering context.

`CGLSetPBuffer` (page 31)
: Attaches a pixel buffer object to a rendering context.

`CGLTexImagePBuffer` (page 33)
: Binds the contents of a pixel buffer to a data source for a texture object.

Getting Error Information

`CGLErrorString` (page 19)
: Returns a string that describes the specified result code.

Getting and Setting Global Information

`CGLSetOption` (page 30)
: Sets the value of a global option.

`CGLGetOption` (page 22)
: Obtains the value of a global option.

`CGLGetVersion` (page 24)
: Gets the major and minor version numbers of the CGL library.

Getting Renderer Information

`CGLDescribeRenderer` (page 15)
: Obtains the value associated with a renderer property.

`CGLDestroyRendererInfo` (page 18)
: Frees resources associated with a renderer information object.

`CGLQueryRendererInfo` (page 26)
: Creates a renderer information object that contains properties and values for all renderers driving the specified displays.

Managing Virtual Screens

`CGLSetVirtualScreen` (page 32)
: Forces subsequent OpenGL commands to the specified virtual screen.

`CGLGetVirtualScreen` (page 24)
: Gets the current virtual screen number associated with rendering context.

Functions

CGLChoosePixelFormat

Creates a pixel format object that satisfies the constraints of the specified buffer and renderer attributes.

```
CGLError CGLChoosePixelFormat (
   const CGLPixelFormatAttribute *attribs,
   CGLPixelFormatObj *pix,
   GLint *npix
);
```

Parameters

attribs

A NULL terminated array that contains a list of buffer and renderer attributes. Attributes can be Boolean or integer. If an attribute is integer, you must supply the desired value immediately following the attribute. If the attribute is Boolean, do not supply a value because its presence in the attributes array implies a true value. For information on the attributes that you can supply, see "Buffer and Renderer Attributes" (page 37) and the Discussion below.

pix

The memory address of a pixel format object. On return, points to a new pixel format object that contains pixel format information and a list of virtual screens. If there are no pixel formats or virtual screens that satisfy the constraints of the buffer and renderer attributes, the value of *pix* is set to NULL.

npix

On return, points to the number of virtual screens referenced by *pix*. If *pix* is NULL, the value of *npix* is set to 0.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

After a pixel format object is created successfully, the integer attributes are set to values that are as close to the desired value as can be provided by the system. Attributes can have different values for each virtual screen. You can use the kCGLPFAMinimumPolicy (page 39) and kCGLPFAMaximumPolicy (page 39) attributes to control how the system chooses the setting. For more information on choosing attributes, see *OpenGL Programming Guide for Mac OS X*.

The Boolean attribute constants include the following:

- kCGLPFAAllRenderers (page 37)
- kCGLPFADoubleBuffer (page 38)
- kCGLPFAStereo (page 38)
- kCGLPFAAuxBuffers (page 38)
- kCGLPFAMinimumPolicy (page 39)
- kCGLPFAMaximumPolicy (page 39)
- kCGLPFAOffScreen (page 39)
- kCGLPFAFullScreen (page 39)
- kCGLPFAAuxDepthStencil (page 39)
- kCGLPFAColorFloat (page 39)

- `kCGLPFAMultisample` (page 39)
- `kCGLPFASupersample` (page 40)
- `kCGLPFASampleAlpha` (page 40)
- `kCGLPFASingleRenderer` (page 40)
- `kCGLPFANoRecovery` (page 40)
- `kCGLPFAAccelerated` (page 41)
- `kCGLPFAClosestPolicy` (page 41)
- `kCGLPFARobust` (page 41)
- `kCGLPFABackingStore` (page 41)
- `kCGLPFAMPSafe` (page 41)
- `kCGLPFAWindow` (page 41)
- `kCGLPFAMultiScreen` (page 42)
- `kCGLPFACompliant` (page 42)
- `kCGLPFAPBuffer` (page 42)
- `kCGLPFARemotePBuffer` (page 42)

The integer attribute constants must be followed by a value:

- `kCGLPFAColorSize` (page 38)
- `kCGLPFAAlphaSize` (page 38)
- `kCGLPFADepthSize` (page 38)
- `kCGLPFAStencilSize` (page 38)
- `kCGLPFAAccumSize` (page 38)
- `kCGLPFASampleBuffers` (page 40)
- `kCGLPFASamples` (page 40)
- `kCGLPFARendererID` (page 40)
- `kCGLPFADisplayMask` (page 42)
- `kCGLPFAVirtualScreenCount` (page 42)

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLDestroyPixelFormat` (page 17)
`CGLDescribePixelFormat` (page 14)

Related Sample Code
GLCarbon1ContextPbuffer
GLCarbonSharedPbuffer
OpenGL Screensaver

Declared In
`OpenGL.h`

CGLClearDrawable

Disassociates a rendering context from any drawable objects attached to it.

```
CGLError CGLClearDrawable (
   CGLContextObj ctx
);
```

Parameters

ctx

A rendering context.

Return Value

A result code. See "CGL Result Codes" (page 56).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLSetOffScreen` (page 29)

`CGLSetFullScreen` (page 28)

Declared In

`OpenGL.h`

CGLCopyContext

Copies the specified state variables from one rendering context to another.

```
CGLError CGLCopyContext (
   CGLContextObj src,
   CGLContextObj dst,
   GLbitfield mask
);
```

Parameters

src

The source rendering context.

dst

The destination rendering context .

mask

A mask that specifies the state variables to copy. Pass a bit field that contains the bitwise `OR` of the state variable names that you want to copy. Use the symbolic mask constants that are passed to the OpenGL function `glPushAttrib`. To copy as many state variables as possible, supply the constant `GL_ALL_ATTRIB_BITS`. For a description of the symbolic mask constants, see OpenGL Reference Manual.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

Not all OpenGL state values can be copied. For example, pixel pack and unpack state, render mode state, and select and feedback state are not copied. The state that can be copied is exactly the state that is manipulated by the OpenGL call `glPushAttrib`.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`OpenGL.h`

CGLCreateContext

Creates a CGL rendering context.

```
CGLError CGLCreateContext (
   CGLPixelFormatObj pix,
   CGLContextObj share,
   CGLContextObj *ctx
);
```

Parameters

`pix`

A pixel format object created by calling the function CGLChoosePixelFormat (page 8).

`share`

The rendering context with which to share the OpenGL object state—including texture objects, programs and shader display lists, vertex array objects, vertex buffer objects, pixel buffer objects, and frame buffer objects—and the object state associated which each of these object types. Pass `NULL` to indicate that no sharing is to take place.

`ctx`

The memory address of a context object. On return, points to a new context object with the buffers and attributes specified by the *`pix`* parameter. If the context can not be created as specified, the value of *`ctx`* is set to `NULL`.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
If the pixel format object you supply is able to support multiple graphics devices, then the rendering context can render transparently across the supported devices. With a multiple device rendering context, sharing is possible only when the relationship between the renderers and the graphics devices they support is the same for all rendering contexts that are shared. Normally you achieve the best display by using the same pixel format object for all shared rendering contexts. For more information, see *OpenGL Programming Guide for Mac OS X*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLSetCurrentContext` (page 27)
`CGLDestroyContext` (page 16)

Related Sample Code
GLCarbon1ContextPbuffer
GLCarbonSharedPbuffer
OpenGL Screensaver

Declared In
`OpenGL.h`

CGLCreatePBuffer

Creates a pixel buffer of the specified size, compatible with the specified texture target.

```
CGLError CGLCreatePBuffer (
   GLsizei width,
   GLsizei height,
   GLenum target,
   GLenum internalFormat,
   GLint max_level,
   CGLPBufferObj *pbuffer
);
```

Parameters

width

The width, in pixels, of the pixel buffer.

height

The height, in pixels, of the pixel buffer.

target

A constant that specifies the type of the pixel buffer target texture. You can supply any of the following texture targets:

- `GL_TEXTURE_2D`, a texture whose dimensions are a power of two.
- `GL_TEXTURE_RECTANGLE_EXT`, a texture whose dimensions are not a power of two.
- `GL_TEXTURE_CUBE_MAP`, a mapped cube texture.

internalFormat

A constant that specifies the internal color format of the pixel buffer, which can be either `GL_RGB` or `GL_RGBA`. The format controls whether the alpha channel of the pixel buffer will be used for texturing operations.

max_level

The maximum level of mipmap detail allowable. Pass `0` for a pixel buffer that is not using mipmaps. The value passed should never exceed the actual maximum number of mipmap levels that can be represented with the given width and height.

pbuffer

On return, points to a new pixel buffer object.

Return Value

A result code. See "CGL Result Codes" (page 56). This function returns `kCGLBadAlloc` if it cannot allocate storage for the pixel buffer data structure. It returns `kCGLBadValue` for any of these conditions:

- A negative *max_level* value provided or a *max_level* value greater than the maximum possible mipmap levels for the given width and height provided.
- A *max_level* value greater than `0` used with a `GL_TEXTURE_RECTANGLE_EXT` texture target
- The dimensions provided for a `GL_TEXTURE_CUBE_MAP` texture target aren't equal.

Discussion

This function does not have any knowledge of OpenGL contexts or pixel format objects and does not specifically allocate the storage needed for the actual pixel buffer. These operations occur when you call the function CGLSetPBuffer (page 31).

You can determine the dimensional limits of a pixel buffer by calling the OpenGL function `glGetInteger`. You can find the maximum size supported by querying `GL_MAX_VIEWPORT_DIMS` and the minimum size by querying `GL_MIN_PBUFFER_VIEWPORT_DIMS_APPLE`, which returns two integer values (similar to `GL_MAX_VIEWPORT_DIMS`). All pixel buffer dimensions that you request with the function `aglCreatePBuffer` should fall within these limits (inclusively) and should comply with any limitations imposed by the texture target you select.

The maximum viewport size supported in Mac OS X is quite large. You should take into consideration the amount of video or system memory required to support the requested pixel buffer size, including additional memory needed for multiple buffers and options such as multisampling.

Availability
Available in Mac OS X v10.3 and later.

See Also
`CGLDestroyPBuffer` (page 16)

Declared In
`OpenGL.h`

CGLDescribePBuffer

Retrieves information that describes the specified pixel buffer object.

```
CGLError CGLDescribePBuffer (
   CGLPBufferObj obj,
   GLsizei *width,
   GLsizei *height,
   GLenum *target,
   GLenum *internalFormat,
   GLint *mipmap
);
```

Parameters

obj
A pointer to pixel buffer object.

width
On return, points to the width, in pixels, of the pixel buffer.

height
On return, points to the height, in pixels, of the pixel buffer.

target
On return, points to a constant that specifies the pixel buffer texture target:

- `GL_TEXTURE_2D`, a texture whose dimensions are a power of two.
- `GL_TEXTURE_RECTANGLE_EXT`, a texture whose dimensions are not a power of two.
- `GL_TEXTURE_CUBE_MAP`, a mapped cube texture.

internalFormat
On return, points to a constant that specifies the internal color format of the pixel buffer—either `GL_RGB` or `GL_RGBA`.

mipmap
On return, points to the mipmap level of the pixel buffer or `0` if it doesn't use mipmaps.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
The width, height, texture target, and internal texture color format of a pixel buffer object are set at its creation and cannot be changed without destroying and recreating the object. The level is set when the pixel buffer object is attached to a rendering context by calling the function CGLSetPBuffer (page 31).

Availability
Available in Mac OS X v10.3 and later.

See Also
`CGLCreatePBuffer` (page 12)

Declared In
`OpenGL.h`

CGLDescribePixelFormat

Retrieves the values of an attribute associated with a pixel format object.

```
CGLError CGLDescribePixelFormat (
   CGLPixelFormatObj pix,
   GLint pix_num,
   CGLPixelFormatAttribute attrib,
   GLint *value
);
```

Parameters

`pix`
: The pixel format object to query.

`pix_num`
: The virtual screen number whose attribute value you want to retrieve. This value must be between `0` and the number of virtual screens minus one.

`attrib`
: The attribute whose value you want to obtain. For a list of possible attributes, see "Buffer and Renderer Attributes" (page 37).

`value`
: On return, points to the value of the attribute.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
A pixel format object can contain different values for each virtual screen, which is why you must supply a virtual screen number in the *`pix_num`* parameter.

You can obtain the number of virtual screens associated with the pixel format object by calling the function `CGLDescribePixelFormat` (page 14), passing the pixel format object, `0` for the virtual screen number, and the attribute constant `kCGLPFAVirtualScreenCount`. For more information about virtual screens, *OpenGL Programming Guide for Mac OS X*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLChoosePixelFormat` (page 8)

Declared In
`OpenGL.h`

CGLDescribeRenderer

Obtains the value associated with a renderer property.

```
CGLError CGLDescribeRenderer (
   CGLRendererInfoObj rend,
   GLint rend_num,
   CGLRendererProperty prop,
   GLint *value
);
```

Parameters

rend
: An opaque renderer information object that contains a description of the renderer capabilities you want to inspect. You can obtain a renderer information object by calling the function CGLQueryRendererInfo (page 26). You must call `CGLDestroyRendererInfo` (page 18) when you no longer need this object.

rend_num
: The index of the renderer inside the renderer information object—a value between 0 and the number of renderers minus one. The number of renderers can be obtained by calling `CGLDescribeRenderer`, passing in *rend*, renderer number 0, and the renderer property `kCGLRPRendererCount`.

prop
: The renderer property whose value you want to obtain. See "Renderer Properties" (page 50) for a list of the constants you can supply for this parameter.

value
: On return, points to the value of the requested property.

Return Value
A result code. See "CGL Result Codes" (page 56).

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLQueryRendererInfo` (page 26)

Related Sample Code
GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

OpenGL Screensaver

Declared In
`OpenGL.h`

CGLDestroyContext

Frees the resources associated with a rendering context.

```
CGLError CGLDestroyContext (
   CGLContextObj ctx
);
```

Parameters

ctx

The rendering context to destroy.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

This function frees all the resources used by the rendering context passed to it. If the rendering context that you pass is the current rendering context, the current context is set to `NULL` and there is no current rendering context after the function executes.

After you call this function, you must make sure that you do not use the destroyed rendering context. This includes using CGL macros in which the rendering context is explicitly passed to OpenGL.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLCreateContext` (page 11)

Related Sample Code

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

OpenGL Screensaver

VideoHardwareInfo

Declared In

`OpenGL.h`

CGLDestroyPBuffer

Releases the resources associated with a pixel buffer object.

```
CGLError CGLDestroyPBuffer (
   CGLPBufferObj pbuffer
);
```

Parameters

pbuffer

The pixel buffer object whose resources you want to release.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion
Call this function only after you no longer need to use the pixel buffer object. Before calling this function, you should delete any texture objects associated with the pixel buffer object. You do not need to make sure that all texturing commands have completed prior to calling this function, because the OpenGL framework manages texturing synchronization.

The results of issuing commands to a destroyed pixel buffer object are undefined.

Availability
Available in Mac OS X v10.3 and later.

See Also
`CGLCreatePBuffer` (page 12)

Declared In
`OpenGL.h`

CGLDestroyPixelFormat

Frees the memory associated with a pixel format object.

```
CGLError CGLDestroyPixelFormat (
   CGLPixelFormatObj pix
);
```

Parameters

`pix`
: The pixel format object to destroy.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
The system makes a copy of the pixel format object when you call the function `CGLCreateContext` (page 11), so you can free a pixel format object immediately after passing it to the context creation function.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLChoosePixelFormat` (page 8)

Related Sample Code
GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

OpenGL Screensaver

VideoHardwareInfo

Declared In
`OpenGL.h`

CGLDestroyRendererInfo

Frees resources associated with a renderer information object.

```
CGLError CGLDestroyRendererInfo (
   CGLRendererInfoObj rend
);
```

Parameters

rend

The renderer information object to destroy.

Return Value

A result code. See "CGL Result Codes" (page 56).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLQueryRendererInfo` (page 26)

`CGLDescribeRenderer` (page 15)

Related Sample Code

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

OpenGL Screensaver

Declared In

`OpenGL.h`

CGLDisable

Disables an option for a rendering context.

```
CGLError CGLDisable (
   CGLContextObj ctx,
   CGLContextEnable pname
);
```

Parameters

ctx

A rendering context.

pname

The option to disable. For a list of possible options, see "Context Options" (page 45).

Return Value

A result code. See "CGL Result Codes" (page 56).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLEnable` (page 19)

`CGLIsEnabled` (page 25)

Related Sample Code
Vertex Optimization

Declared In
`OpenGL.h`

CGLEnable

Enables an option for a rendering context.

```
CGLError CGLEnable (
   CGLContextObj ctx,
   CGLContextEnable pname
);
```

Parameters

`ctx`
A rendering context.

`pname`
The option to enable. For a list of possible options, see "Context Options" (page 45).

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
Some context options have values associated with them. Use `CGLSetParameter` (page 30) and `CGLGetParameter` (page 22) to set and get context parameter values.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLDisable` (page 18)
`CGLIsEnabled` (page 25)

Related Sample Code
QTCoreVideo102
QTCoreVideo103
QTCoreVideo201
QTCoreVideo301
VertexPerformanceDemo

Declared In
`OpenGL.h`

CGLErrorString

Returns a string that describes the specified result code.

```
const char * CGLErrorString (
   CGLError error
);
```

Parameters

error

The CGL result code constant returned from a CGL function. For a description of these constants, see "CGL Result Codes" (page 56).

Return Value

An error string that describes the result code constant passed in the *error* parameter. If the result code is invalid, returns the string "No such error code."

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenGL.h`

CGLFlushDrawable

Copies the back buffer of a double-buffered context to the front buffer.

```
CGLError CGLFlushDrawable (
   CGLContextObj ctx
);
```

Parameters

ctx

The context object.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

To create a double-buffered context, specify the `kCGLPFADoubleBuffer` attribute (see "Buffer and Renderer Attributes" (page 37)) when you create the pixel format object for the rendering context. If the backing store attribute is set to `false` the buffers may be exchanged rather than copied. This is often the case in full-screen mode. If the receiver is not a double-buffered context, this call does nothing.

If you set the swap interval attribute (`kCGLCPSwapInterval`) appropriately, the copy takes place during the vertical retrace of the monitor, rather than immediately after `CGLFlushDrawable` is called. An implicit `glFlush` is performed by `CGLFlushDrawable` before it returns. For optimal performance, an application should not call `glFlush` immediately before calling `CGLFlushDrawable`. Subsequent OpenGL commands can be issued immediately after calling `CGLFlushDrawable`, but are not executed until the buffer copy is completed. For more information about `kCGLCPSwapInterval`, see "Context Parameters" (page 46).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OpenGL.h`

CGLGetCurrentContext

Returns the current rendering context.

```
CGLContextObj CGLGetCurrentContext (
   void
);
```

Return Value
The current rendering context. If there is none, returns `NULL`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLSetCurrentContext` (page 27)

Related Sample Code
GLCarbonSharedPbuffer

GLSLShowpiece

NSOpenGL Fullscreen

QTCoreVideo201

SurfaceVertexProgram

Declared In
`CGLCurrent.h`

CGLGetOffScreen

Retrieves an offscreen buffer and its parameters for a specified rendering context.

```
CGLError CGLGetOffScreen (
   CGLContextObj ctx,
   GLsizei *width,
   GLsizei *height,
   GLint *rowbytes,
   void **baseaddr
);
```

Parameters

ctx
: A rendering context.

width
: On return, points to the width, in pixels, of the offscreen buffer. If the rendering context is not attached to an offscreen drawable object, the value of *width* is set to `0`.

height
: On return, points to the height, in pixels, of the offscreen buffer. If the rendering context is not attached to an offscreen drawable object, the value of *height* is set to `0`.

rowbytes
: On return, points to the number of bytes per row of the offscreen buffer. If the context is not attached to an offscreen drawable object, the value of *rowbytes* is set to `0`.

baseaddr

On return, points to the base address of the offscreen buffer. If the context is not attached to an offscreen drawable object, the value of *baseaddr* is set to `NULL`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLSetOffScreen` (page 29)

Declared In
`OpenGL.h`

CGLGetOption

Obtains the value of a global option.

```
CGLError CGLGetOption (
   CGLGlobalOption pname,
   GLint *param
);
```

Parameters

pname

The name of the option whose value you want to get. See "Global Options" (page 48) for a list of constants you can pass.

param

On return, a pointer to the value of the option.

Return Value
A result code. See "CGL Result Codes" (page 56).

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLSetOption` (page 30)

Declared In
`OpenGL.h`

CGLGetParameter

Retrieves the value of a rendering context parameter.

```
CGLError CGLGetParameter (
   CGLContextObj ctx,
   CGLContextParameter pname,
   GLint *params
);
```

Parameters

ctx

A rendering context.

pname

The parameter whose value you want to retrieve. For a list of possible parameters, see "Context Parameters" (page 46).

params

On return, points to the value of the parameter.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

Some parameters may need to have a corresponding context option enabled for their value to take effect. You can enable, disable, and test whether an option is enabled with `CGLEnable` (page 19), `CGLDisable` (page 18), and `CGLIsEnabled` (page 25).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLSetParameter` (page 30)

Related Sample Code

GLSLShowpiece

NSOpenGL Fullscreen

Declared In

`OpenGL.h`

CGLGetPBuffer

Retrieves a pixel buffer and its parameters for a specified rendering context.

```
CGLError CGLGetPBuffer (
   CGLContextObj ctx,
   CGLPBufferObj *pbuffer,
   GLenum *face,
   GLint *level,
   GLint *screen
);
```

Parameters

ctx

A rendering context.

pbuffer

On return, points to the pixel buffer object attached to the rendering context.

face

On return, points to the cube map face that is set if the pixel buffer texture target type is `GL_TEXTURE_CUBE_MAP`; otherwise `0` for all other texture target types.

level

On return, points to the current mipmap level for drawing.

screen

On return, points to the current virtual screen number, as set by the last valid call to CGLSetPBuffer (page 31).

Return Value
A result code. See "CGL Result Codes" (page 56).

Availability
Available in Mac OS X v10.3 and later.

See Also
`CGLSetPBuffer` (page 31)

Declared In
`OpenGL.h`

CGLGetVersion

Gets the major and minor version numbers of the CGL library.

```
void CGLGetVersion (
   GLint *majorvers,
   GLint *minorvers
);
```

Parameters

`majorvers`
: On return, points to he major version number of the CGL library.

`minorvers`
: On return, points to the minor version number of the CGL library.

Discussion
CGL implementations with the same major version number are upwardly compatible, meaning that the implementation with the highest minor number is a superset of the version with the lowest minor number.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenGL.h`

CGLGetVirtualScreen

Gets the current virtual screen number associated with rendering context.

```
CGLError CGLGetVirtualScreen (
   CGLContextObj ctx,
   GLint *screen
);
```

Parameters

`ctx`
: A rendering context.

`screen`
: On return, points to the virtual screen associated with the context. The value is always `0` on a single-monitor system and `-1` if the function fails for any reason.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

The current virtual screen can change when a drawable object is moved or resized across graphics device boundaries. A change in the current virtual screen can affect the return values of some OpenGL functions and in most cases also means that the renderer has changed.

For detailed information on virtual screens, see *OpenGL Programming Guide for Mac OS X*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLGetVirtualScreen` (page 24)

Declared In

`OpenGL.h`

CGLIsEnabled

Reports whether an option is enabled for a rendering context.

```
CGLError CGLIsEnabled (
   CGLContextObj ctx,
   CGLContextEnable pname,
   GLint *enable
);
```

Parameters

`ctx`

A rendering context.

`pname`

The option to query. For a list of possible options, see "Context Options" (page 45).

`enable`

On return, *`enable`* is set to `true` if the option is enabled.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

To set or get parameter values associated with a context option, use `CGLSetParameter` (page 30) or `CGLGetParameter` (page 22).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLEnable` (page 19)

`CGLDisable` (page 18)

Declared In

`OpenGL.h`

CGLLockContext

Locks a CGL rendering context.

```
CGLError CGLLockContext (
   CGLContextObj ctx
);
```

Parameters

ctx

The CGL context to lock.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

The function `CGLLockContext` blocks the thread it is on until all other threads have unlocked the same context using the function `CGLUnlockContext`. You can use `CGLLockContext` recursively. Context-specific CGL calls by themselves do not require locking, but you can guarantee serial processing for a group of calls by surrounding them with `CGLLockContext` and `CGLUnlockContext`. Keep in mind that calls from the OpenGL API (the API provided by the Architecture Review Board) require locking.

Applications that use NSOpenGL classes with multithreading can lock contexts using the functions `CGLLockContext` and `CGLUnlockContext`. To perform rendering in a thread other than the main one, you can lock the context that you want to access and safely execute OpenGL commands. The locking calls must be placed around all OpenGL calls in all threads.

For more information on multithreading OpenGL applications, see *OpenGL Programming Guide for Mac OS X*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`CGLUnlockContext` (page 34)

Related Sample Code

OpenGLCaptureToMovie

VideoViewer

Declared In

`OpenGL.h`

CGLQueryRendererInfo

Creates a renderer information object that contains properties and values for all renderers driving the specified displays.

```
CGLError CGLQueryRendererInfo (
   GLuint display_mask,
   CGLRendererInfoObj *rend,
   GLint *nrend
);
```

Parameters

display_mask

A bit field that contains the bitwise `OR` of OpenGL display masks returned by the `CGDisplayIDToOpenGLDisplayMask` function. If you want to obtain information for all renderers in the system, set every bit in *display_mask* to `true`.

rend

The memory address of a renderer information object. On return, points to a renderer information object that describes all renderers that are able to drive the displays specified by the *display_mask* parameter. If *display_mask* does not specify any displays, the value of *rend* is set to `NULL`. You must call `CGLDestroyRendererInfo` (page 18) when you no longer need this object.

nrend

On return, points to the number of renderers described in the renderer information object. If *display_mask* does not specify any displays, the value of *nrend* is set to `0`.

Return Value

A result code. See "CGL Result Codes" (page 56).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLDescribeRenderer` (page 15)

`CGLDestroyRendererInfo` (page 18)

Related Sample Code

GLCarbon1ContextPbuffer

GLCarbonSharedPbuffer

OpenGL Screensaver

Declared In

`OpenGL.h`

CGLSetCurrentContext

Sets the specified rendering context as the current rendering context.

```
CGLError CGLSetCurrentContext (
   CGLContextObj ctx
);
```

Parameters

ctx

The rendering context to set as the current rendering context. Pass `NULL` to release the current rendering context without assigning a new one.

Return Value

A result code. See "CGL Result Codes" (page 56). If the function fails, the current context remains unchanged.

Discussion
There can be only one current rendering context. Subsequent OpenGL rendering calls operate on the current rendering context to modify the drawable object associated with it.

You can use AGL macros to bypass the current rendering context mechanism and maintain your own current rendering context.

A context is current on a per-thread basis. Multiple threads must serialize calls into the same context.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLGetCurrentContext` (page 21)

Related Sample Code
CALayerEssentials
GLCarbon1ContextPbuffer
GLCarbonSharedPbuffer
OpenGL Screensaver
VideoHardwareInfo

Declared In
`CGLCurrent.h`

CGLSetFullScreen

Attaches a rendering context to its full-screen drawable object.

```
CGLError CGLSetFullScreen (
   CGLContextObj ctx
);
```

Parameters

`ctx`
A rendering context.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
Before calling this function, you must set up the rendering context using a pixel format object created with the `kCGLPFAFullScreen` attribute (see "Buffer and Renderer Attributes" (page 37)). Some OpenGL renderers, such as the software renderer, do not support full-screen mode. After you call the function CGLChoosePixelFormat (page 8) with the full-screen attribute, you need to check whether the pixel format object is created successfully.

You must capture the display prior to entering full-screen mode and release it after exiting. After calling this function subsequent OpenGL drawing is rendered into the entire screen. For more information, see *OpenGL Programming Guide for Mac OS X*.

To exit full-screen mode, call `CGLClearDrawable` (page 9).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OpenGL.h`

CGLSetOffScreen

Attaches a rendering context to an offscreen buffer.

```
CGLError CGLSetOffScreen (
   CGLContextObj ctx,
   GLsizei width,
   GLsizei height,
   GLint rowbytes,
   void *baseaddr
);
```

Parameters

`ctx`
A rendering context.

`width`
The width, in pixels, of the offscreen buffer.

`height`
The height, in pixels, of the offscreen buffer.

`rowbytes`
The number of bytes per row of the offscreen buffer, which must be greater than or equal to *`width`* times bytes per pixel.

`baseaddr`
A pointer to a block of memory to use as the offscreen buffer. The size of the memory must be at least *`rowbytes`***`height`* bytes.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
Before calling this function, you must set up the rendering context using a pixel format object created with the `kCGLPFAOffScreen` attribute. For more information about `kCGLPFAOffScreen`, see "Buffer and Renderer Attributes" (page 37).

After calling this function subsequent OpenGL drawing is rendered into the offscreen buffer and the viewport of the rendering context is set to the full size of the offscreen area.

To exit offscreen mode call `CGLClearDrawable` (page 9).

To obtain functionality similar to offscreen mode on renderers that do not support it, attach the context to a hidden window and use the OpenGL function `glReadPixels`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLGetOffScreen` (page 21)

Declared In
`OpenGL.h`

CGLSetOption

Sets the value of a global option.

```
CGLError CGLSetOption (
   CGLGlobalOption pname,
   GLint param
);
```

Parameters

pname
The name of the option whose value you want to set. See "Global Options" (page 48) for a list of constants you can pass.

param
The value to set the option to.

Return Value
A result code. See "CGL Result Codes" (page 56).

Discussion
This function changes the values of options that affect the operation of OpenGL in all rendering contexts in the application, not just the current rendering context.

Availability
Available in Mac OS X v10.0 and later.

See Also
`CGLGetOption` (page 22)

Declared In
`OpenGL.h`

CGLSetParameter

Sets the value of a rendering context parameter.

```
CGLError CGLSetParameter (
   CGLContextObj ctx,
   CGLContextParameter pname,
   const GLint *params
);
```

Parameters

ctx
A rendering context.

pname
The parameter whose value you want to set. For a list of possible parameters, see "Context Parameters" (page 46).

params

A pointer to the value to the parameter to.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

Some parameters may need to have a corresponding context option enabled for their value to take effect. You can enable, disable, and test whether an option is enabled with `CGLEnable` (page 19), `CGLDisable` (page 18), and `CGLIsEnabled` (page 25).

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLGetParameter` (page 22)

Related Sample Code

NSOpenGL Fullscreen

NURBSSurfaceVertexProg

SurfaceVertexProgram

VBL

Declared In

`OpenGL.h`

CGLSetPBuffer

Attaches a pixel buffer object to a rendering context.

```
CGLError CGLSetPBuffer (
   CGLContextObj ctx,
   CGLPBufferObj pbuffer,
   GLenum face,
   GLint level,
   GLint screen
);
```

Parameters

ctx

A rendering context.

pbuffer

A pixel buffer object.

face

The cube map face to draw if the pixel buffer texture target type is `GL_TEXTURE_CUBE_MAP`; otherwise pass `0`.

level

The mipmap level to draw. This must not exceed the maximum mipmap level set when the pixel buffer object was created. Pass `0` for a texture target that does not support mipmaps.

screen

A virtual screen value. The virtual screen determines the renderer OpenGL uses to draw to the pixel buffer object. For best performance, for a pixel buffer used as a texture source, you should supply the a virtual screen value that results in using the same renderer used by the context that's the texturing target.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

The first time you call this function for a specific pixel buffer object, the system creates the necessary buffers. The buffers are created to support the attributes dictated by the pixel format object used to create the rendering context and by the parameters used to create the pixel buffer object. The storage requirements for pixel buffer objects, which can be quite large, are very similar to the requirements for windows or views with OpenGL contexts attached. All drawable objects compete for the same scarce resources. This function can fail is there is not enough contiguous VRAM for each buffer. It's best to code defensively with a scheme that reduces resource consumption without causing the application to resort to failure. Unless, of course, failure is the only viable alternative.

The ability to attach a pbuffer to a context is supported only on renderers that export `GL_APPLE_pixel_buffer` in the `GL_EXTENSIONS` string. Before calling this function, you should programmatically determine if it's possible to attach a pbuffer to a context by querying `GL_EXTENSIONS` in the context and looking for `GL_APPLE_pixel_buffer`. If that extension is not present, the renderer won't allow setting the pbuffer.

In order of performance, these are the renderers you should consider using when setting up a rendering context to attach to a pbuffer:

- A hardware renderer.
- The generic render, but only with an offscreen pixel format and `glTexSubImage`.
- The Apple software renderer, which supports pbuffers in Mac OS X v10.4.8 and later.

Availability

Available in Mac OS X v10.3 and later.

See Also

`CGLGetPBuffer` (page 23)

Declared In

`OpenGL.h`

CGLSetVirtualScreen

Forces subsequent OpenGL commands to the specified virtual screen.

```
CGLError CGLSetVirtualScreen (
   CGLContextObj ctx,
   GLint screen
);
```

Parameters

ctx

A rendering context.

`screen`

A virtual screen number, which must be a value between `0` and the number of virtual screens minus one. The number of virtual screens available in a context can be obtained by calling the function `CGLDescribePixelFormat` (page 14), passing in the pixel format object used to create the rendering context, `0` for the virtual screen number (*`pix_num`* parameter), and the attribute constant `kCGLPFAVirtualScreenCount`.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

Setting the virtual screen forces the renderer associated with the virtual screen to process OpenGL commands issued to the specified context. Changing the virtual screen changes the current renderer. You should use this function only when it is necessary to override the default behavior. The current virtual screen is normally set automatically. Because the current virtual screen determines which OpenGL renderer is processing commands, the return values of all `glGetXXX` functions can be affected by the current virtual screen.

For detailed information on virtual screens, see *OpenGL Programming Guide for Mac OS X*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`CGLGetVirtualScreen` (page 24)

Declared In

`OpenGL.h`

CGLTexImagePBuffer

Binds the contents of a pixel buffer to a data source for a texture object.

```
CGLError CGLTexImagePBuffer (
   CGLContextObj ctx,
   CGLPBufferObj pbuffer,
   GLenum source
);
```

Parameters

`ctx`

A rendering context, which is the target context for the texture operation. This is the context that you plan to render content to. This is not the context attached to the pixel buffer.

`pbuffer`

A pixel buffer object.

`source`

The source buffer to texture from, which should be a valid OpenGL buffer such as `GL_FRONT` or `GL_BACK` and should be compatible with the buffer and renderer attributes that you used to create the rendering context attached to the pixel buffer. This means that the pixel buffer must possess the buffer in question for the texturing operation to succeed.

Return Value

A result code. See "CGL Result Codes" (page 56).

Discussion

You must generate and bind a texture name (using standard OpenGL texturing calls) that is compatible with the pixel buffer texture target. Don't supply a texture object that was used previously for nonpixel buffer texturing operations unless you first call `glDeleteTextures` to regenerate the texture name.

If you modify the content of a pixel buffer that uses mipmap levels, you must call this function again before drawing with the pixel buffer, to ensure that the content is synchronized with OpenGL. For pixel buffers without mipmaps, simply rebind to the texture object to synchronize content.

No OpenGL texturing calls that modify a pixel buffer texture content are permitted (such as `glTexSubImage2D` or `glCopyTexImage2D`) with the pixel buffer texture as the destination. It *is* permitted to use texturing commands to read data *from* a pixel buffer texture, such as `glCopyTexImage2D`, with the pixel buffer texture as the *source*. It is also legal to use OpenGL functions such as `glReadPixels` to read the contents of a pixel buffer directly through the pixel buffer context.

Note that texturing with the `CGLTexImagePBuffer` function can fail to produce the intended results without error in the same way other OpenGL texturing commands can normally fail. The function fails if you set an incompatible filter mode, do not enable the proper texture target, or other conditions described in the OpenGL specification.

You don't need to share a context to use a pixel buffer object as a texture source. You can use independent pixel format objects and OpenGL contexts for both the pixel buffer and the target drawable object without sharing resources, and still texture using a pixel buffer in the target context.

For details on how to use a pixel buffer object as a texture source, see *OpenGL Programming Guide for Mac OS X*.

Availability

Available in Mac OS X v10.3 and later.

See Also

`CGLCreatePBuffer` (page 12)

`CGLSetPBuffer` (page 31)

Declared In

`OpenGL.h`

CGLUnlockContext

Unlocks a CGL rendering context.

```
CGLError CGLUnlockContext (
   CGLContextObj ctx
);
```

Parameters

`ctx`

The CGL context to unlock.

Return Value

A result code. See "CGL Result Codes" (page 56).

Availability

Available in Mac OS X v10.4 and later.

See Also
`CGLLockContext` (page 26)

Related Sample Code
OpenGLCaptureToMovie
VideoViewer

Declared In
`OpenGL.h`

Data Types

CGLContextObj

Represents a pointer to an opaque CGL context object.

```
typedef struct _CGLContextObject *CGLContextObj;
```

Discussion
This data type points to a structure that CGL uses to maintain state and other information associated with an OpenGL rendering context. Use the functions described in "Managing Contexts" (page 5) and "Getting and Setting Context Options" (page 6) to create, manage, access, and free a CGL context object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

CGLPixelFormatObj

Represents a pointer to an opaque pixel format object.

```
typedef struct _CGLPixelFormatObject *CGLPixelFormatObj;
```

Discussion
This data type points to a structure that CGL uses to maintain pixel format and virtual screen information for a given set of renderer and buffer options. Use the functions described in "Managing Pixel Format Objects" (page 5) to create, manage, access, and free a pixel format object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

CGLRendererInfoObj

Represents a pointer to an opaque renderer information object.

```
typedef struct _CGLRendererInfoObject *CGLRendererInfoObj;
```

Discussion
This data type points to a structure that CGL uses to maintain information about the renderers associated with a display. Use the functions described in "Getting Renderer Information" (page 7) to create, access, and free a renderer information object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

CGLPBufferObj

Represents a pointer to an opaque pixel buffer object.

```
typedef struct _CGLPBufferObject *CGLPBufferObj;
```

Discussion
This data type points to a structure that CGL uses for hardware accelerated offscreen drawing. Use the functions described in "Managing Pixel Format Objects" (page 5) to create, manage, access, and free a pixel buffer object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Constants

Buffer Mode Flags

Define constants used to set buffer modes.

```
#define kCGLMonoscopicBit   0x00000001
#define kCGLStereoscopicBit 0x00000002
#define kCGLSingleBufferBit 0x00000004
#define kCGLDoubleBufferBit 0x00000008
```

Constants

`kCGLMonoscopicBit`
Specifies to use a left buffer.

`kCGLStereoscopicBit`
Specifies to a left and right buffer.

`kCGLSingleBufferBit`
Specifies to use a front buffer.

`kCGLDoubleBufferBit`
Specifies to use a front and back buffer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Buffer and Renderer Attributes

Specify attributes used to choose pixel formats and virtual screens.

```
typedef enum _CGLPixelFormatAttribute {
    kCGLPFAAllRenderers       =   1,
    kCGLPFADoubleBuffer       =   5,
    kCGLPFAStereo             =   6,
    kCGLPFAAuxBuffers         =   7,
    kCGLPFAColorSize          =   8,
    kCGLPFAAlphaSize          =  11,
    kCGLPFADepthSize          =  12,
    kCGLPFAStencilSize        =  13,
    kCGLPFAAccumSize          =  14,
    kCGLPFAMinimumPolicy      =  51,
    kCGLPFAMaximumPolicy      =  52,
    kCGLPFAOffScreen          =  53,
    kCGLPFAFullScreen         =  54,
    kCGLPFASampleBuffers      =  55,
    kCGLPFASamples            =  56,
    kCGLPFAAuxDepthStencil    =  57,
    kCGLPFAColorFloat         =  58,
    kCGLPFAMultisample        =  59,
    kCGLPFASupersample        =  60,
    kCGLPFASampleAlpha        =  61,
    kCGLPFARendererID         =  70,
    kCGLPFASingleRenderer     =  71,
    kCGLPFANoRecovery         =  72,
    kCGLPFAAccelerated        =  73,
    kCGLPFAClosestPolicy      =  74,
    kCGLPFARobust             =  75,
    kCGLPFABackingStore       =  76,
    kCGLPFAMPSafe             =  78,
    kCGLPFAWindow             =  80,
    kCGLPFAMultiScreen        =  81,
    kCGLPFACompliant          =  83,
    kCGLPFADisplayMask        =  84,
    kCGLPFAPBuffer            =  90,
    kCGLPFARemotePBuffer      =  91,
    kCGLPFAVirtualScreenCount = 128,
} CGLPixelFormatAttribute;
```

Constants

`kCGLPFAAllRenderers`

This constant is a Boolean attribute. If it is present in the attributes array, pixel format selection is open to all available renderers, including debug and special-purpose renderers that are not OpenGL compliant. Do not supply a value with this constant because its presence in the array implies `true`.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLPFADoubleBuffer`
: This constant is a Boolean attribute. If it is present in the attributes array, only double-buffered pixel formats are considered. Otherwise, only single-buffered pixel formats are considered. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAStereo`
: This constant is a Boolean attribute. If it is present in the attributes array, only stereo pixel formats are considered. Otherwise, only monoscopic pixel formats are considered. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAAuxBuffers`
: The associated value is a nonnegative integer that indicates the desired number of auxiliary buffers. Pixel formats with the smallest number of auxiliary buffers that meet or exceeds the specified number are preferred.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAColorSize`
: The associated value is a nonnegative buffer size specification. A color buffer that most closely matches the specified size is preferred. If unspecified, OpenGL chooses a color size that matches the screen.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAAlphaSize`
: The associated value is a nonnegative buffer size specification. An alpha buffer that most closely matches the specified size is preferred.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFADepthSize`
: The associated value is a nonnegative depth buffer size specification. A depth buffer that most closely matches the specified size is preferred.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAStencilSize`
: The associated value is a nonnegative integer that indicates the desired number of stencil bitplanes. The smallest stencil buffer of at least the specified size is preferred.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAAccumSize`
: The associated value is a nonnegative buffer size specification. An accumulation buffer that most closely matches the specified size is preferred.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAMinimumPolicy`
: This constant is a Boolean attribute. If it is present in the attributes array, the pixel format choosing policy is altered for the color, depth, and accumulation buffers such that only buffers of size greater than or equal to the desired size are considered. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAMaximumPolicy`
: This constant is a Boolean attribute. If it is present in the attributes array, the pixel format choosing policy is altered for the color, depth, and accumulation buffers such that, if a nonzero buffer size is requested, the largest available buffer is preferred. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAOffScreen`
: This constant is a Boolean attribute. If it is present in the attributes array, only renderers that are capable of rendering to an off-screen memory area and have buffer depth exactly equal to the desired buffer depth are considered. The `kCGLPFAClosestPolicy` attribute is implied. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAFullScreen`
: This constant is a Boolean attribute. If it is present in the attributes array, only renderers that are capable of rendering to a full-screen drawable object are considered. The `kCGLPFASingleRenderer` attribute is implied. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAAuxDepthStencil`
: This constant is a Boolean attribute. If it is present in the attributes array, each auxiliary buffer has its own depth-stencil buffer. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.2 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAColorFloat`
: This constant is a Boolean attribute. If it is present in the attributes array, color buffers store floating-point pixels. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.2 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAMultisample`
: This constant is a Boolean attribute. If it is present in the attributes array, specifies a hint to the driver to prefer multisampling. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLPFASupersample`
: This constant is a Boolean attribute. If it is present in the attributes array, specifies a hint to the driver to prefer supersampling. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLPFASampleAlpha`
: This constant is a Boolean attribute. If it is present in the attributes array, request alpha filtering when multisampling. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLPFASampleBuffers`
: The number of multisample buffers. The associated value is a nonnegative integer that indicates the number of existing independent sample buffers. Typically, the value is 0 if no multi-sample buffer exists or 1. This attribute is not useful in the attribute array.

  Available in Mac OS X v10.1 and later.

  Declared in `CGLTypes.h`.

`kCGLPFASamples`
: The number of samples per multisample buffer. The associated value is a nonnegative integer that indicates the desired number of samples that can be taken within a single pixel. The smallest sample buffer with at least the specified number of samples is preferred.

  Available in Mac OS X v10.1 and later.

  Declared in `CGLTypes.h`.

`kCGLPFARendererID`
: The associated value is a nonnegative renderer ID number and can be any of the constants defined in "Renderer IDs" (page 49). OpenGL renderers that match the specified ID are preferred. Of note is `kCGLRendererGenericID` which selects the Apple software renderer. The other constants select renderers for specific hardware vendors.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFASingleRenderer`
: This constant is a Boolean attribute. If it is present in the attributes array, a single rendering engine is chosen. On systems with multiple screens, this disables ability of OpenGL to drive different monitors through different graphics accelerator cards with a single context. This attribute is not generally useful. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFANoRecovery`
: This constant is a Boolean attribute. If it is present in the attributes array, the OpenGL failure recovery mechanisms are disabled. Normally, if an accelerated renderer fails due to lack of resources, OpenGL automatically switches to another renderer. This attribute disables these features so that rendering is always performed by the chosen renderer. This attribute is not generally useful. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAAccelerated`
: This constant is a Boolean attribute. If it is present in the attributes array, only hardware accelerated renderers are considered. If false, accelerated renderers are still preferred. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAClosestPolicy`
: This constant is a Boolean attribute. If it is present in the attributes array, the pixel format choosing policy is altered for the color buffer such that the buffer closest to the requested size is preferred, regardless of the actual color buffer depth of the supported graphics device. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFARobust`
: This constant is a Boolean attribute. If it is present in the attributes array, only renderers that do not have any failure modes associated with a lack of video card resources are considered. This attribute is not generally useful. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFABackingStore`
: This constant is a Boolean attribute. If it is present in the attributes array, OpenGL only considers renderers that have a back color buffer the full size of the drawable object and that guarantee the back buffer contents to be valid after a call to `CGLFlushDrawable` (page 20). Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAMPSafe`
: This constant is a Boolean attribute. If it is present in the attributes array, OpenGL only considers renderers that are thread-safe. Because all renderers are thread-safe, this attribute is not useful. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAWindow`
: This constant is a Boolean attribute. If it is present in the attributes array, only renderers that are capable of rendering to a window are considered. This attribute is implied if neither `kCGLPFAFullScreen` nor `kCGLPFAOffScreen` is specified. Because CGL only supports full-screen of off-screen drawable objects, this attribute is not useful. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAMultiScreen`
: This constant is a Boolean attribute. If it is present in the attributes array, only renderers capable of driving multiple screens are considered. This attribute is not generally useful. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFACompliant`
: This constant is a Boolean attribute. If it is present in the attributes array, pixel format selection is only open to OpenGL compliant renderers. This attribute is implied unless `kCGLPFAAllRenderers` is specified. This attribute is not useful in the attribute array. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFADisplayMask`
: The associated value is a bit mask of supported physical screens. All screens specified in the bit mask are guaranteed to be supported by the pixel format. Screens not specified in the bit mask may still be supported. The bit mask is managed by the Quartz Display Services, available in the `CGDirectDisplay.h` header of the Application Services umbrella framework. A `CGDirectDisplayID` must be converted to an OpenGL display mask using the function `CGDisplayIDToOpenGLDisplayMask`. This attribute is not generally useful.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAPBuffer`
: This constant is a Boolean attribute. If it is present in the attributes array, format can be used to render to a pixel buffer. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLPFARemotePBuffer`
: This constant is a Boolean attribute. If it is present in the attributes array, format can be used to render offline to a pixel buffer. Do not supply a value with this constant because its presence in the array implies `true`.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLPFAVirtualScreenCount`
: This attribute may be used to obtain the number of virtual screens specified by an existing pixel format object. To retrieve the value, call the function `CGLDescribePixelFormat` (page 14), passing the pixel format object, the virtual screen number `0`, and this attribute. This attribute is not useful in the attribute array that's used to create a pixel format object.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

Discussion

These constants are used by CGLChoosePixelFormat (page 8) and CGLDescribePixelFormat (page 14). The existence of a Boolean attribute in the attribute array of `CGLChoosePixelFormat` implies a `true` value. Other attribute constants must be followed by a value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Color and Accumulation Buffer Format Flags

Specify formats for the color and accumulation buffers.

```
#define kCGLRGB444Bit       0x00000040
#define kCGLARGB4444Bit     0x00000080
#define kCGLRGB444A8Bit     0x00000100
#define kCGLRGB555Bit       0x00000200
#define kCGLARGB1555Bit     0x00000400
#define kCGLRGB555A8Bit     0x00000800
#define kCGLRGB565Bit       0x00001000
#define kCGLRGB565A8Bit     0x00002000
#define kCGLRGB888Bit       0x00004000
#define kCGLARGB8888Bit     0x00008000
#define kCGLRGB888A8Bit     0x00010000
#define kCGLRGB101010Bit    0x00020000
#define kCGLARGB2101010Bit  0x00040000
#define kCGLRGB101010_A8Bit 0x00080000
#define kCGLRGB121212Bit    0x00100000
#define kCGLARGB12121212Bit 0x00200000
#define kCGLRGB161616Bit    0x00400000
#define kCGLRGBA16161616Bit 0x00800000
#define kCGLRGBFloat64Bit   0x01000000
#define kCGLRGBAFloat64Bit  0x02000000
#define kCGLRGBFloat128Bit  0x04000000
#define kCGLRGBAFloat128Bit 0x08000000
#define kCGLRGBFloat256Bit  0x10000000
#define kCGLRGBAFloat256Bit 0x20000000
```

Constants

`kCGLRGB444Bit`
Specifies a format that has 16 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=11:8, G=7:4, B=3:0.

`kCGLARGB4444Bit`
Specifies a format that has 16 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=15:12, R=11:8, G=7:4, B=3:0.

`kCGLRGB444A8Bit`
Specifies a format that has 8-16 bits per pixel with an RGB channel layout, and the channels located in the following bits: A=7:0, R=11:8, G=7:4, B=3:0.

`kCGLRGB555Bit`
Specifies a format that has 16 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=14:10, G=9:5, B=4:0.

`kCGLARGB1555Bit`
Specifies a format that has 16 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=15, R=14:10, G=9:5, B=4:0.

`kCGLRGB555A8Bit`
: Specifies a format that has 8-16 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=7:0, R=14:10, G=9:5, B=4:0.

`kCGLRGB565Bit`
: Specifies a format that has 16 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=15:11, G=10:5, B=4:0.

`kCGLRGB565A8Bit`
: Specifies a format that has 6-16 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=7:0, R=15:11, G=10:5, B=4:0.

`kCGLRGB888Bit`
: Specifies a format that has 32 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=23:16, G=15:8, B=7:0.

`kCGLARGB8888Bit`
: Specifies a format that has 32 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=31:24, R=23:16, G=15:8, B=7:0.

`kCGLRGB888A8Bit`
: Specifies a format that has 8-32 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=7:0, R=23:16, G=15:8, B=7:0.

`kCGLRGB101010Bit`
: Specifies a format that has 32 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=29:20, G=19:10, B=9:0.

`kCGLARGB2101010Bit`
: Specifies a format that has 32 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=31:30 R=29:20, G=19:10, B=9:0.

`kCGLRGB101010_A8Bit`
: Specifies a format that has 8-32 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=7:0 R=29:20, G=19:10, B=9:0.

`kCGLRGB121212Bit`
: Specifies a format that has 48 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=35:24, G=23:12, B=11:0.

`kCGLARGB12121212Bit`
: Specifies a format that has 48 bits per pixel with an ARGB channel layout, and the channels located in the following bits: A=47:36, R=35:24, G=23:12, B=11:0.

`kCGLRGB161616Bit`
: Specifies a format that has 64 bits per pixel with an RGB channel layout, and the channels located in the following bits: R=63:48, G=47:32, B=31:16.

`kCGLRGBA16161616Bit`
: Specifies a format that has 64 bits per pixel with an ARGB channel layout, and the channels located in the following bits: R=63:48, G=47:32, B=31:16, A=15:0.

`kCGLRGBFloat64Bit`
: Specifies a format that has 64 bits per pixel with an RGB half floating-point channel layout.

`kCGLRGBAFloat64Bit`
: Specifies a format that has 64 bits per pixel with an ARGB half floating-point channel layout.

`kCGLRGBFloat128Bit`
: Specifies a format that has 128 bits per pixel with an RGB IEEE floating-point channel layout.

`kCGLRGBAFloat128Bit`
: Specifies a format that has 128 bits per pixel with an ARGB IEEE floating-point channel layout.

`kCGLRGBFloat256Bit`
: Specifies a format that has 256 bits per pixel with an RGB IEEE double channel layout.

`kCGLRGBAFloat256Bit`
: Specifies a format that has 256 bits per pixel with an ARGB IEEE double channel layout.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Context Options

Specify options that affect a rendering context.

```
typedef enum _CGLContextEnable {
    kCGLCESwapRectangle    = 201,
    kCGLCERasterization    = 221,
    kCGLCEStateValidation  = 301,
    kCGLCESurfaceBackingSize = 305,
    kCGLCEDisplayListOptimization = 307
} CGLContextEnable;
```

Constants

`kCGLCESwapRectangle`

If enabled, the area of the drawable object that is affected by `CGLFlushDrawable` (page 20) is restricted to a rectangle specified by the values of `kCGLCPSwapRectangle`. However, the portion of the drawable object that lies outside of the swap rectangle may still be flushed to the screen by a visibility change or other user interface action. To set or get the values of `kCGLCPSwapRectangle`, use the functions `CGLSetParameter` (page 30) or `CGLGetParameter` (page 22), respectively. For more information about `kCGLCPSwapRectangle`, see "Context Parameters" (page 46).

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLCERasterization`

If disabled, all rasterization of 2D and 3D primitives is disabled. This state is useful for debugging and to characterize the performance of an OpenGL driver without actually rendering.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLCEStateValidation`

If enabled, OpenGL inspects the context state each time that `CGLSetVirtualScreen` (page 32) is called to ensure that it is in an appropriate state for switching between renderers. Normally, the state is inspected only when it is actually necessary to switch renderers. In CGL, a renderer is switched only if you call `CGLSetVirtualScreen` (page 32) with a virtual screen number different than the current one.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLCESurfaceBackingSize`

If enabled, overrides the surface backing size.

Available in Mac OS X v10.3 and later.

Declared in `CGLTypes.h`.

`kCGLCEDisplayListOptimization`

If disabled, turns off optimization for the display list.

Available in Mac OS X v10.3 and later.

Declared in `CGLTypes.h`.

Discussion

These are used by the functions CGLEnable (page 19), CGLDisable (page 18), and CGLIsEnabled (page 25).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGLTypes.h`

Context Parameters

Specify parameters that apply to a specific rendering context.

```
typedef enum _CGLContextParameter {
    kCGLCPSwapRectangle    = 200,
    kCGLCPSwapInterval     = 222,
    kCGLCPDispatchTableSize = 224,
    kCGLCPClientStorage    = 226,
    kCGLCPSurfaceTexture    = 228,
    kCGLCPSurfaceOrder      = 235,
    kCGLCPSurfaceOpacity    = 236,
    kCGLCPSurfaceBackingSize = 304,
    kCGLCPSurfaceSurfaceVolatile = 306,
    kCGLCPReclaimResources  = 308,
    kCGLCPCurrentRendererID  = 309,
    kCGLCPGPUVertexProcessing  = 310,
    kCGLCPGPUFragmentProcessing  = 311
} CGLContextParameter;
```

Constants

`kCGLCPSwapRectangle`

Set or get the swap rectangle. The swap rectangle is represented as an array of four `long` values: `{x, y, width, height}`. For this rectangle to affect the outcome of calling the function `CGLFlushDrawable` (page 20), the context option `kCGLCESwapRectangle` must be enabled. For more information about `kCGLCESwapRectangle`, see "Context Options" (page 45).

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLCPSwapInterval`

Set or get the swap interval. The swap interval is represented as one `long` value. If the swap interval is set to `0` (the default), `CGLFlushDrawable` (page 20) executes as soon as possible, without regard to the vertical refresh rate of the monitor. If the swap interval is set to `1`, the buffers are swapped only during the vertical retrace of the monitor.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLCPDispatchTableSize`
: Set or get the dispatch table size.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLCPClientStorage`
: Set or get an arbitrary 32-bit value. A typical usage would be to store a pointer to application-specific data associated with the context.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLCPSurfaceTexture`
: Set the surface texture. Supply a surface ID, target, and internal format.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLCPSurfaceOrder`
: Set or get the position of the OpenGL surface relative to the window. A value of `1` means that the position is above the window; a value of `-1` specifies a position that is below the window.

  Available in Mac OS X v10.2 and later.

  Declared in `CGLTypes.h`.

`kCGLCPSurfaceOpacity`
: Set or get the surface opacity. A value of `1` means the surface is opaque (the default); `0` means completely transparent.

  Available in Mac OS X v10.2 and later.

  Declared in `CGLTypes.h`.

`kCGLCPSurfaceBackingSize`
: Set or get the height and width of the back buffer. You can use this to let the system scale an image automatically on swap to a variable size buffer. The back buffer size remains fixed at the size that you set up regardless of whether the image is resized to display larger onscreen.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLCPSurfaceSurfaceVolatile`
: Set or get the volatile state of a surface.

  Available in Mac OS X v10.3 and later.

  Declared in `CGLTypes.h`.

`kCGLCPReclaimResources`
: Enable or disable reclaiming resources.

  Available in Mac OS X v10.4 and later.

  Declared in `CGLTypes.h`.

`kCGLCPCurrentRendererID`
: The current renderer ID. You can get this setting.

  Available in Mac OS X v10.4 and later.

  Declared in `CGLTypes.h`.

`kCGLCPGPUVertexProcessing`
: The GPU is currently processing vertices with the GPU. You can get this state.

  Available in Mac OS X v10.4 and later.

  Declared in `CGLTypes.h`.

`kCGLCPGPUFragmentProcessing`
: The CPU is currently processing fragments with the GPU. You can get this state.

  Available in Mac OS X v10.4 and later.

  Declared in `CGLTypes.h`.

Discussion
These constants are used by the functions CGLSetParameter (page 30) and CGLGetParameter (page 22).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Global Options

Specify options that apply globally.

```
typedef enum _CGLGlobalOption {
    kCGLGOFormatCacheSize  = 501,
    kCGLGOClearFormatCache = 502,
    kCGLGORetainRenderers  = 503,
    kCGLGOResetLibrary     = 504,
    kCGLGOUseErrorHandler  = 505,
} CGLGlobalOption;
```

Constants
`kCGLGOFormatCacheSize`
: Set or get the pixel format cache size, a positive integer. After an application calls `CGLChoosePixelFormat` (page 8) for the last time, it may set the cache size to `1` to minimize the memory used by CGL. If an application intends to use `n` different attribute lists to choose `n` different pixel formats repeatedly, then the application should set the cache size to `n` to maximize performance. The cache size is initially set to `5`.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLGOClearFormatCache`
: If set to a `true` value, the pixel format object cache contents are freed. This does not affect the size of the cache for future storage of pixel format objects. To minimize the memory consumed by the cache, the application should also set the cache size to `1` via the `kCGLGOFormatCacheSize` global option. `CGLGetOption` (page 22) always reports a `false` value for this constant.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLGORetainRenderers`
: If `true`, CGL does not unload any plug-in renderers even if they are no longer in use. This is useful to improve the performance of applications that repeatedly destroy and recreate their only (or last) rendering context. Normally, when the last context created by a particular plug-in renderer is destroyed, that renderer is unloaded from memory. If `false`, CGL returns to its normal mode of operation and all renderers that are not in use are unloaded.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLGOResetLibrary`
: If set to a `true` value, CGL is reset to its initial state. All contexts created with `CGLCreateContext` (page 11) are destroyed, all plug-in renderers are unloaded from memory, and global options are reset to their initial values. Renderer information objects and pixel format objects are not destroyed. `CGLGetOption` (page 22) always reports a `false` value for this constant.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLGOUseErrorHandler`
: If `true`, CGL errors are propagated to Core Graphics.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

Discussion
These constants are used by the functions CGLSetOption (page 30) and CGLGetOption (page 22).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Renderer IDs

Define constants that specify hardware and software renderers.

```
kCGLRendererGenericID         0x00020200
kCGLRendererGenericFloatID    0x00020400
kCGLRendererAppleSWID         0x00020600
kCGLRendererATIRage128ID      0x00021000
kCGLRendererATIRadeonID       0x00021200
kCGLRendererATIRageProID      0x00021400
kCGLRendererATIRadeon8500ID   0x00021600
kCGLRendererATIRadeon9700ID   0x00021800
kCGLRendererATIRadeonX1000ID  0x00021900
kCGLRendererGeForce2MXID      0x00022000
kCGLRendererGeForce3ID        0x00022200
kCGLRendererGeForceFXID       0x00022400
kCGLRendererVTBladeXP2ID      0x00023000
kCGLRendererIntel900ID        0x00024000
kCGLRendererMesa3DFXID        0x00040000
```

Constants
`kCGLRendererGenericID`
: Specifies the software renderer. Deprecated on Intel-based Macintosh computers.

`kCGLRendererGenericFloatID`
: Specifies the floating-point software renderer.

`kCGLRendererAppleSWID`
: Specifies the Apple software renderer ID.

`kCGLRendererATIRage128ID`
: Specifies the ATI Rage128 renderer.

`kCGLRendererATIRadeonID`
: Specifies the ATI Radeon renderer.

`kCGLRendererATIRageProID`
: Specifies the ATI RagePro renderer.

`kCGLRendererATIRadeon8500ID`
: Specifies the ATI Radeon 8500 renderer.

`kCGLRendererATIRadeon9700ID`
: Specifies the ATI Radeon 9700 renderer.

`kCGLRendererATIRadeonX1000ID`
: Specifies the ATI Radio X1000 renderer.

`kCGLRendererGeForce2MXID`
: Specifies the NVIDIA GeForce2MX renderer.

`kCGLRendererGeForce3ID`
: Specifies the NVIDIA GeForce3 renderer.

`kCGLRendererGeForceFXID`
: Specifies the NVIDIA GeForceFX renderer.

`kCGLRendererVTBladeXP2ID`
: Specifies the VTBook renderer.

`kCGLRendererIntel900ID`
: Specifies the Intel GMA 900 renderer.

`kCGLRendererMesa3DFXID`
: Specifies the Mesa 3DFX renderer.

Declared In
`CGLRenderers.h`

Renderer Properties

Specify renderer properties.

```
typedef enum _CGLRendererProperty {
    kCGLRPOffScreen           =  53,
    kCGLRPFullScreen          =  54,
    kCGLRPRendererID          =  70,
    kCGLRPAccelerated         =  73,
    kCGLRPRobust              =  75,
    kCGLRPBackingStore        =  76,
    kCGLRPMPSafe              =  78,
    kCGLRPWindow              =  80,
    kCGLRPMultiScreen         =  81,
    kCGLRPCompliant           =  83,
    kCGLRPDisplayMask         =  84,
    kCGLRPBufferModes         = 100,
    kCGLRPColorModes          = 103,
    kCGLRPAccumModes          = 104,
    kCGLRPDepthModes          = 105,
    kCGLRPStencilModes        = 106,
    kCGLRPMaxAuxBuffers       = 107,
    kCGLRPMaxSampleBuffers    = 108,
    kCGLRPMaxSamples          = 109,
    kCGLRPSampleModes         = 110,
    kCGLRPSampleAlpha         = 111,
    kCGLRPVideoMemory         = 120,
    kCGLRPTextureMemory       = 121,
    kCGLRPRendererCount       = 128,
} CGLRendererProperty;
```

Constants

`kCGLRPOffScreen`

If `true`, the renderer supports offscreen drawable objects.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPFullScreen`

If `true`, the renderer supports full screen drawable objects.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPRendererID`

The associated value is the renderer ID. Renderer ID constants are associated with specific hardware vendors. See "Renderer IDs" (page 49).

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPAccelerated`

If `true`, the renderer is hardware accelerated.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPRobust`

If `true`, the renderer does not have any failure modes caused by a lack of video card resources.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPBackingStore`
: If `true`, the renderer can provide a back color buffer the full size of the drawable object and can guarantee the back buffer contents to be valid after a call to `CGLFlushDrawable` (page 20).

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPMPSafe`
: If `true`, the renderer is thread-safe. All renderers are thread-safe in Mac OS X.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPWindow`
: If `true`, the renderer supports window drawable objects.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPMultiScreen`
: If `true`, the renderer is presently attached to multiple displays.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPCompliant`
: If `true`, the renderer is OpenGL compliant. All renderers are OpenGL compliant in Mac OS X.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPDisplayMask`
: The associated value is a bit mask of physical displays that the renderer can drive. The bit mask is managed by Quartz Display Services. A `CGDirectDisplayID` data type must be converted to an OpenGL display mask using the function `CGDisplayIDToOpenGLDisplayMask`. For more information on this function, see *Quartz Display Services Reference*.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPBufferModes`
: The associated value is the bitwise `OR` of buffer mode flags supported by the renderer. The value can be any of the constants defined in "Buffer Mode Flags" (page 36).

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPColorModes`
: The associated value is the bitwise `OR` of color format flags supported by the renderer. The value can be any of the constants defined in "Color and Accumulation Buffer Format Flags" (page 43).

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPAccumModes`
: The associated value is the bitwise `OR` of color/accumulation buffer format flags supported by the renderer. The value can be any of the constants defined in "Color and Accumulation Buffer Format Flags" (page 43).

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

`kCGLRPDepthModes`
The associated value is the bitwise `OR` of depth/stencil buffer depth flags supported by the renderer. The value can be any of the constants defined in "Stencil and Depth Modes" (page 54).

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPStencilModes`
The associated value is the bitwise `OR` of depth/stencil buffer depth flags supported by the renderer. The value can be any of the constants defined in "Stencil and Depth Modes" (page 54).

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPMaxAuxBuffers`
The associated value is the maximum number of auxiliary buffers supported by the renderer.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPMaxSampleBuffers`
The associated value is the maximum number of independent sample buffers supported by the renderer. Typically, the value is `0` if no multisample buffer exists, or `1` if one exists.

Available in Mac OS X v10.1 and later.

Declared in `CGLTypes.h`.

`kCGLRPMaxSamples`
The associated value is the maximum number of samples per pixel that the renderer supports.

Available in Mac OS X v10.1 and later.

Declared in `CGLTypes.h`.

`kCGLRPSampleModes`
A bit field of supported sample modes.

Available in Mac OS X v10.3 and later.

Declared in `CGLTypes.h`.

`kCGLRPSampleAlpha`
If `true`, there is support for alpha sampling.

Available in Mac OS X v10.3 and later.

Declared in `CGLTypes.h`.

`kCGLRPVideoMemory`
The associated value is the number of bytes of video memory available to the renderer.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPTextureMemory`
The associated value is the number of bytes of texture memory available to the renderer.

Available in Mac OS X v10.0 and later.

Declared in `CGLTypes.h`.

`kCGLRPRendererCount`
: The associated value is the number of renderers in a specific renderer information object. To determine the number of renderers in a renderer information object, call the function `CGLDescribeRenderer` (page 15), passing in the object, renderer number `0`, and this renderer property.

  Available in Mac OS X v10.0 and later.

  Declared in `CGLTypes.h`.

Discussion
These constants are used by the function CGLDescribeRenderer (page 15).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Sampling Modes

Define modes used for full scene anti-aliasing.

```
#define kCGLSupersampleBit 0x00000001
#define kCGLMultisampleBit 0x00000002
```

Constants

`kCGLSupersampleBit`
: Specifies supersampling.

`kCGLMultisampleBit`
: Specifies multisampling.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Stencil and Depth Modes

Define resolutions for the depth and stencil buffers.

```
#define kCGL0Bit            0x00000001
#define kCGL1Bit            0x00000002
#define kCGL2Bit            0x00000004
#define kCGL3Bit            0x00000008
#define kCGL4Bit            0x00000010
#define kCGL5Bit            0x00000020
#define kCGL6Bit            0x00000040
#define kCGL8Bit            0x00000080
#define kCGL10Bit           0x00000100
#define kCGL12Bit           0x00000200
#define kCGL16Bit           0x00000400
#define kCGL24Bit           0x00000800
#define kCGL32Bit           0x00001000
#define kCGL48Bit           0x00002000
#define kCGL64Bit           0x00004000
#define kCGL96Bit           0x00008000
#define kCGL128Bit          0x00010000
```

Constants

`kCGL0Bit`
: Specifies a 0-bit resolution.

`kCGL1Bit`
: Specifies a 1-bit resolution.

`kCGL2Bit`
: Specifies a 2-bit resolution.

`kCGL3Bit`
: Specifies a 3-bit resolution.

`kCGL4Bit`
: Specifies a 4-bit resolution.

`kCGL5Bit`
: Specifies a 5-bit resolution.

`kCGL6Bit`
: Specifies a 6-bit resolution.

`kCGL8Bit`
: Specifies a 8-bit resolution.

`kCGL10Bit`
: Specifies a 10-bit resolution.

`kCGL12Bit`
: Specifies a 12-bit resolution.

`kCGL16Bit`
: Specifies a 16-bit resolution.

`kCGL24Bit`
: Specifies a 24-bit resolution.

`kCGL32Bit`
: Specifies a 32-bit resolution.

`kCGL48Bit`
: Specifies a 48-bit resolution.

`kCGL64Bit`
: Specifies a 64-bit resolution.

`kCGL96Bit`
  Specifies a 96-bit resolution.

`kCGL128Bit`
  Specifies a 128-bit resolution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CGLTypes.h`

Result Codes

The following result code constants, declared in the CGLTypes.h header file, can be used as parameters to the function `CGLErrorString` (page 19).

Result Code	Value	Description
`kCGLNoError`	0	No error. Available in Mac OS X v10.3 and later.
`kCGLBadAttribute`	10000	Invalid pixel format attribute. Valid attributes can be found in "Buffer and Renderer Attributes" (page 37). Available in Mac OS X v10.0 and later.
`kCGLBadProperty`	10001	Invalid renderer property. Valid renderer properties can be found in "Renderer Properties" (page 50). Available in Mac OS X v10.0 and later.
`kCGLBadPixelFormat`	10002	Invalid pixel format object. A valid pixel format object can be obtained by calling the function `CGLChoosePixelFormat` (page 8). Available in Mac OS X v10.0 and later.
`kCGLBadRendererInfo`	10003	Invalid renderer information object. A valid renderer information object can be obtained by calling the function `CGLQueryRendererInfo` (page 26). Available in Mac OS X v10.0 and later.
`kCGLBadContext`	10004	Invalid context object. A valid context object can be obtained by calling the function `CGLCreateContext` (page 11). Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`kCGLBadDrawable`	10005	Invalid drawable object. This error occurs when you attempt to attach a second, incompatible, rendering context to a drawable object. For more information about incompatible contexts, see the discussion section of the function `CGLCreateContext` (page 11). This error also occurs when you attempt to attach a context to a full-screen drawable object, and the color depth of the drawable object is different than that specified by the pixel format object used to create the context. The `kCGLPFAColorSize` attribute, described in "Buffer and Renderer Attributes" (page 37), specifies the color depth of a pixel format. Available in Mac OS X v10.0 and later.
`kCGLBadDisplay`	10006	Invalid display. Available in Mac OS X v10.0 and later.
`kCGLBadState`	10007	Invalid context state. This error occurs when a context state is inspected for readiness to switch renderers. To be in a valid state, a context be in render mode and have an attribute stack depth of `0`, and a modelview, projection, and texture stack depth of `1`. For more information about state verification, see the context enable constant `kCGLCEStateValidation`, described in "Context Options" (page 45). Available in Mac OS X v10.0 and later.
`kCGLBadValue`	10008	Invalid numerical value. Available in Mac OS X v10.0 and later.
`kCGLBadMatch`	10009	Invalid share context. Two contexts are a bad match if their pixel format objects use different renderers. Available in Mac OS X v10.0 and later.
`kCGLBadEnumeration`	10010	Invalid constant. Available in Mac OS X v10.0 and later.
`kCGLBadOffScreen`	10011	Invalid offscreen drawable object. Available in Mac OS X v10.0 and later.
`kCGLBadFullScreen`	10012	Invalid full-screen drawable object. Available in Mac OS X v10.0 and later.
`kCGLBadWindow`	10013	Invalid window. Available in Mac OS X v10.0 and later.
`kCGLBadAddress`	10014	Invalid memory address. This error occurs when you pass an invalid pointer into a function that requires a memory address other than `NULL`. Available in Mac OS X v10.0 and later.
`kCGLBadCodeModule`	10015	Invalid code module. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`kCGLBadAlloc`	10016	Invalid memory allocation. This error occurs when CGL is unable to allocate memory. Available in Mac OS X v10.0 and later.
`kCGLBadConnection`	10017	Invalid connection to Core Graphics. Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *CGL Reference*.

Date	Notes
2007-06-28	Added more information about renderer information objects.
	Updated `CGLChoosePixelFormat` (page 8) and "`Buffer and Renderer Attributes`" (page 37).
	Revised the texture target information in `CGLCreatePBuffer` (page 12).
	Revised the discussion in `CGLSetPBuffer` (page 31).
	Added `kCGLCPCurrentRendererID` (page 47), `kCGLCPGPUVertexProcessing` (page 48), and `kCGLCPGPUFragmentProcessing` (page 48).
	Updated the renderers listed in "Renderer IDs" (page 49).
2006-07-07	Minor noncontent change.
2006-07-24	Changed the title from "CGL Framework Reference."
2006-05-23	Updated for Mac OS X v10.4.
	Added the functions CGLLockContext (page 26) and CGLUnlockContext (page 34).
	Added "Sampling Modes" (page 54).
	Added documentation for "Buffer Mode Flags" (page 36), "Color and Accumulation Buffer Format Flags" (page 43), "Renderer IDs" (page 49), and "Stencil and Depth Modes" (page 54). Most of these constants were previously embedded in the discussion of other constants. There are several new color accumulation buffer format flags.
	Removed documentation for internal formats because these are part of the OpenGL specification. They are not part of the CGL API. Instead see the OpenGL Reference Manual and the OpenGL Programming Guide, which are both published by Addison-Wesley.
	Added constants to "Context Options" (page 45), "Context Parameters" (page 46), and "CGL Result Codes" (page 56).
	Revised "Introduction" (page 5).

Date	Notes
	Edited content to make it more consistent with other Apple OpenGL documentation.
	Fixed formatting.
	Added availability information.
	Updated header file information.
	Added See Also sections.
2005-11-09	Fixed links, added header file information, and removed old boilerplate text from the Introduction.

Index